**PLAYWRIGHT**

Playwright is an open-source automation library for web browsers developed by Microsoft and primarily used for automating interactions with web browsers like Chromium, Firefox, and WebKit-based browsers. It provides high-level APIs for browser automation, making it easy to perform actions such as clicking buttons, filling out forms, and navigating between pages.

It also supports advanced features like [intercepting network requests](https://www.lambdatest.com/blog/cypress-intercept/), emulating mobile devices, and taking screenshots. Playwright is an [end-to-end testing](https://www.lambdatest.com/learning-hub/end-to-end-testing) framework slowly gaining popularity in the market.

**Why Playwright**

* Cross-browser support
* Speed and reliability
* Trace Viewer
* CodeGen
* Community support

**Features of Playwright**

Playwright is a powerful automation tool for web applications that provides developers and QA engineers with a wide range of features. Here are some of the key features of Playwright:

* **Cross-language**

Playwright is a cross-language [automation testing tool](https://www.lambdatest.com/blog/automation-testing-tools/) for web applications. It supports automation in several programming languages, such as JavaScript, TypeScript, Python, Java, and C#. This cross-language support allows developers and QA engineers to use their preferred programming language and tools to write tests.

* **Cross-browser/platform**

Playwright supports all modern rendering engines, including Chromium, WebKit, and Firefox. It is a cross-platform automation tool for web applications. It supports automation on Windows, macOS, and Linux operating systems, making running tests on different environments easy.

* **Auto-wait**

One of its key features is the auto-wait functionality, which allows Playwright to automatically wait for elements to be ready before performing actions on them. Auto-wait is enabled by default in Playwright and can be customized using the setDefaultTimeout method of the Playwright class. This method sets the default timeout for waiting for an element to appear or become visible.

* **Screenshot and Video Capture**

Playwright provides APIs for capturing screenshots and recording videos of web pages, allowing you to validate the state of web pages during automation tests visually.

* **Assertion**

Playwright is compatible with various testing frameworks that provide built-in assertion libraries, such as Jest, Mocha, and Jasmine. These frameworks allow you to write and execute tests that include assertions on various conditions, such as element existence, text content, visibility, and more.

* **Codegen**

Codegen is used in Playwright to generate code for various programming languages such as JavaScript, Python, and C#. Codegen is useful when working with Playwright because it can generate code that follows best practices and reduces the amount of manual coding required.

* **Playwright Inspector**

Playwright Inspector is a feature of the Playwright automation library that allows you to debug and troubleshoot your Playwright scripts. It provides an interactive graphical user interface (GUI) that lets you inspect the state of the browser and the page, interact with page elements, and debug your scripts.

* **Reporting**

Playwright provides built-in reporters that you can use to generate reports for your tests. [Playwright reporters](https://www.lambdatest.com/learning-hub/playwright-reporting) help you to visualize and understand the results of your tests in a better way. Here are the reporters available in Playwright: ‘list’, ’dot’, ’line’, ’junit’, and ‘HTML’.

**Limitations of Playwright**

* Playwright is new, and it’s still evolving—scope for improvement.
* Playwright doesn’t support Native Mobile Apps

**Playwright Documentation**

* Link : <https://playwright.dev/> Go through this documentation for more information

**Installation**

* Go to the terminal & type npm init playwright@latest
* Choose between TypeScript or JavaScript (default is TypeScript)
* Name of your Tests folder (default is tests or e2e if you already have a tests folder in your project)
* Add a GitHub Actions workflow to easily run tests on CI
* Install Playwright browsers (default is true)

**What’s installed**

* playwright.config.js
* package.json
* package-lock.json
* tests/example.spec.js
* tests-examples/demo-todo-app.spec.js

**playwright.config.js:** As the name indicates the configuration file is the place where we configure the execution behaviour of the playwright. The configuration file is typically named playwright.config.js(/ts).

Using the configuration file, we can configure headless mode, full screen, screenshot options, baseURL, browser options, etc.

The Playwright can be used with different test runners such as Mocha, Jasmine, Jest, etc. Similar way playwright has its own test runner called the playwright test. The playwright test is the default test runner for the playwright.

**Locators in Playwright**

* page.getByText() : Find the element that matches the given text
* page.getByRole(): Find the element that matches the role attribute
* page.getByLabel(): Find the element that matches the label
* page.getByTestId(): Find the element that matches the data-testid attribute
* page.getByTitle(): Find the element that matches the title attribute
* page.locator(<css> or <xpath>): Find the element by using the CSS or XPath

**Assertions**

The Playwright supports soft and hard assertions both.

There are many assertions which expect library provides some of them are

## Auto-retrying assertions

| **Assertion** | **Description** |
| --- | --- |
| [await expect(locator).toBeChecked()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-be-checked) | Checkbox is checked |
| [await expect(locator).toBeDisabled()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-be-disabled) | Element is disabled |
| [await expect(locator).toBeEnabled()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-be-enabled) | Element is enabled |
| [await expect(locator).toBeFocused()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-be-focused) | Element is focused |
| [await expect(locator).toBeVisible()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-be-visible) | Element is visible |
| [await expect(locator).toContainText()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-contain-text) | Element contains text |
| [await expect(locator).toHaveAttribute()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-have-attribute) | Element has a DOM attribute |
| [await expect(locator).toHaveCount()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-have-count) | List has exact number of children |
| [await expect(locator).toHaveText()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-have-text) | Element matches text |
| [await expect(locator).toHaveValue()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-have-value) | Input has a value |
| [await expect(locator).toHaveValues()](https://playwright.dev/docs/api/class-locatorassertions#locator-assertions-to-have-values) | Select has options selected |
| [await expect(page).toHaveTitle()](https://playwright.dev/docs/api/class-pageassertions#page-assertions-to-have-title) | Page has a title |
| [await expect(page).toHaveURL()](https://playwright.dev/docs/api/class-pageassertions#page-assertions-to-have-url) | Page has a URL |

## Non-retrying assertions

| **Assertion** | **Description** |
| --- | --- |
| [expect(value).toBe()](https://playwright.dev/docs/api/class-genericassertions#generic-assertions-to-be) | Value is the same |
| [expect(value).toBeCloseTo()](https://playwright.dev/docs/api/class-genericassertions#generic-assertions-to-be-close-to) | Number is approximately equal |
| [expect(value).toBeGreaterThan()](https://playwright.dev/docs/api/class-genericassertions#generic-assertions-to-be-greater-than) | Number is more than |
| [expect(value).toBeGreaterThanOrEqual()](https://playwright.dev/docs/api/class-genericassertions#generic-assertions-to-be-greater-than-or-equal) | Number is more than or equal |

## Negating matchers

## In general, we can expect the opposite to be true by adding a not to the front of the matchers:

## Examples:

## expect(value).not.toEqual(0);

## await expect(locator).not.toContainText('some text');

## Soft assertions

By default, when the assertions fail the test terminates, but if we use the soft assertions do not terminate the execution, but the test will be marked as failed at the end.

The Playwright provides a command

expect.soft() for soft assertions

Example:

expect.soft(page.locator('#title').toHaveText('Javascript')

## Methods

## The page.goto() is used for navigating to a specific URL.

## The page.locator(‘ ’).fill('javascript') is used to enter text.

## The page.locator(‘ ’).click() is used to perform click action

## The page.locator(‘ ’).hover() to mouse hover on element

## The page.locator(‘ ’).check('javascript') is used to check the checkbox

## The page.locator (‘ ’).uncheck('javascript') is used to uncheck the checkbox

## The page.frameLocator('#courses-iframe') to locate frame.

## The page.locator(‘ ’).allInnerTexts() returns text of all the elements

The page.locator(‘ ’).**screenshot**({ path: 'filename'}) takes element screenshoot

The page.**screenshot**({path: 'filename',fullpage:true}) takes full page screenshoot

The page.locator(‘ ’).**selectOption**(‘Value’) to handle the dropdown

The page.**textContent**(‘locator’) to get element text

The page.**pause**() to pause the execution.

**Handling Alert in Playwright**

* Dialog popups are native to the browser or operating systems. The dialogs need special mechanism to handle as you cannot inspect the locator for these pop-ups.
* There are different types of pop ups such as alert(), confirm(), prompt()

**Click on Ok**

page.on('dialog', dialog => dialog.accept());

**Click on Cancel**

page.on('dialog', dialog => dialog.dismiss ());

**Handling Prompt**

page.on('dialog', dialog => dialog.accept("RSAcademy"));

### Some of the Important command line options

* **Run all the tests**

npx playwright test

* **Run a single test file**

npx playwright test tests/todo-single.spec.ts

* **Run multiple tests**

npx playwright test tests/todo-page/ tests/landing-page/

* **Run tests in headed mode**

npx playwright test --headed

* **Run tests on Specific browser**

npx playwright test --browser "chromium"

* **Retry failed test**

npx playwright test --retries 2